Trabalho Intermediário

Victor Marcius Magalhães Pinto

1 Descrição da Tarefa

teste

2 Implementação de funções intermediárias

Para a execução da tarefa, algumas funções auxiliares foram implementadas. A implementação das mesmas pode ser visto a seguir.

```
> checkAcc <- function(result, correct) {
+
+     if (length(result) != length(correct)){
+         cat("[checkAcc Error] Sizes of result and correct are diferent! Please
+         return(NULL)
+     }
+
+     count <- 0
+     for (i in seq(length(result))) {
+         if (result[i] == correct[i]) count <- count + 1
+     }
+
+     percCorrect <- count / length(result) * 100
+
+     return(c(count, percCorrect))
+ }
> MostraImagem <- function( x )
+ {
+
+     img <- matrix( x, nrow=64 )
+     cor <- rev( gray(50:1/50) )
+     image( rotate( img ), col=cor )
+ }
> pdfnvar <- function(x,m,K,n) {
+
+     y <- ((1/(((2*pi)^n*(det(K)))))*exp(-0.5*(t(x-m)%*%(pseudoinverse(K))%*%(x-
+     return(y)
```

```
+
+ }
```

3 Execução do código

Primeiramente, foram carragadas as imagens contidas no pacote de faces. O pacote contém 40 classes de imagens, com 10 amostras cada.

3.1 Carregando os dados

```
> data( faces )
> faces <- t( faces )
> rotate <- function(x) t( apply(x, 2, rev) )
> y <- NULL
> for(i in 1:nrow(faces) )
+ {
+     y <- c( y, ((i-1) %/% 10) + 1 )
+ }
> # Nomeando os atributos
> nomeColunas <- NULL
> for(i in 1:ncol(faces) )
+ {
+     nomeColunas <- c(nomeColunas, paste("a", as.character(i), sep=".") )
+ }
> nomeLinhas <- NULL
> for(i in 1:nrow(faces)) {
+     nomeLinhas <- c(nomeLinhas, paste("face", as.character(y[i]),as.character(i
+ }
> colnames(faces) <- nomeColunas
> faces <- as.data.frame(faces, row.names = nomeLinhas)
> rm(nomeColunas)
> rm(nomeLinhas)
```

Cada imagem foi transformada, de uma matrix de 64x64 pixels, em um vetor de 4096 features, e inserida em um dataframe de 400 linhas.

3.2 Diminuindo o número de atributos com PCA

Em seguida, foi realizada a diminuição do número de atributos da base de dados utilizando o algoritmo do PCA. Em implementações anteriores, foi utilizada a

função *preProcess* juntamente com a *predict* do pacote *caret*, o que nos informva o número mínimo de atributos para uma exatidão de 95% de 123 features. Porém, o pacote apresentou problemas de funcionamento em dias posteriores, o que nos exigiu a utilização da função *prcomp*, que pode ser vista no trecho de código a serguir.

```
> facesPCAaux <- prcomp(faces, center=TRUE, retx=TRUE, scale=TRUE)
> facesPCA <- facesPCAaux$x[,1:123]
>
```

Para a implementação do algoritmo utilizado, será feito uso da quantidade de features mínimas que haviam sido apontadas anteriormente pelo algoritmo pre-Process, do pacote *caret*, de 123 features.

3.3 Diminuindo o número de atributos com MDS

O mesmo procedimento, e a mesma limitação do número de features foi realizado para o me´todo MDS. O algoritmo pode ser visto a seguir.

```
> kMDS <- 123
> facesMDS <- cmdscale(dist(faces), k=kMDS)
>
```

3.4 Gerando sets de treino e teste

Como dito anteriormente, para o treinamento e classificação pelos algoritmos, foram utilizadas 5 amostras de cada face. A geração das amostras pdoe ser vista no trecho de código a seguir. A mesma possui uma alta complexidade de implementação, visto da utilizazão de loops internos, e de toda o algoritmo, é o trecho que consome mais tempo de execução

```
> dim_classe <- 10
> numClasses <- 400
> numAmostras <- 10
> seqN <- sample(numAmostras)
> porcAmostTrain <- 0.5
> N <- seqN[1:(porcAmostTrain*numAmostras)]
> nSamplesTrain <- length(N)
> n <- seqN[(porcAmostTrain*numAmostras+1):numAmostras]
> nSamplesTest <- length(n)
> xtreino <- c()
```

```
> xtreinoPCA <- c()
> xtreinoMDS <- c()
> ytreino <- c()
> xteste <- c()
> xtestePCA <- c()
> xtesteMDS <- c()
> yteste <- c()
> for(r in seq(1,numClasses,numAmostras)) {
+     for(i in N) {
+         xtreino <- rbind(xtreino, (faces[r+i-1,]))
+         xtreinoPCA <- rbind(xtreinoPCA, (facesPCA[r+i-1,]))
+         xtreinoMDS <- rbind(xtreinoMDS, (facesMDS[r+i-1,]))
+         ytreino <- c(ytreino,(y[r+i-1]))
+     }
+
+     for(i in n) {
+         xteste <- rbind(xteste, (faces[r+i-1,]))
+         xtestePCA <- rbind(xtestePCA, (facesPCA[r+i-1,]))
+         xtesteMDS <- rbind(xtesteMDS, (facesMDS[r+i-1,]))
+         yteste <- c(yteste,(y[r+i-1]))
+     }
+ }
>
```

3.5 Classificando com KNN e PCA

Começando então à realizar as classificações, foi feito uso da função *knn* para a classificação das amostras de teste, com base nos resultados oferecidos pelas amostras de treinamento. Como a função nos permite escolher o número de vizinhos a ser considerados para o correto estabelecimento, foram relizadas 10 rotinas de treinamento, variando esse parâmetro entre 1 e 10. A rotina utilizada pode ser vista a seguir.

```
> resultKNNPCA <- c()
> for (i in seq(10)) {
+     separation <- knn(xtreinoPCA,xtestePCA,ytreino,k=i)
+     resultKNNPCA <- rbind(resultKNNPCA,matrix(c(i,(checkAcc(separation, yteste)
+ }
> colnames(resultKNNPCA) <- c("N","Accuracy")
> meanResultKNNPCA <- mean(resultKNNPCA[,2])
```

```
> results[["KNNPCA"]] <- resultKNNPCA
>
```

Os resultados da execução deste algoritmo podem ser vistos a seguir.

```
       N Accuracy
 [1,]  1     84.0
 [2,]  2     76.0
 [3,]  3     72.5
 [4,]  4     71.5
 [5,]  5     69.5
 [6,]  6     70.0
 [7,]  7     67.0
 [8,]  8     63.0
 [9,]  9     62.5
[10,] 10     63.0
```

3.6 Classificando com KNN e MDS

Realizando a mesma classificação anterior, agora porém para os dados reduzidos através de MDS. O algoritimo utilizado é idêntico ao anterior, como pode ser visto a seguir.

```
> resultKNNMDS <- c()
> for (i in seq(10)) {
+     separation <- knn(xtreinoMDS,xtesteMDS,ytreino,k=i)
+     resultKNNMDS <- rbind(resultKNNMDS,matrix(c(i,(checkAcc(separation, yteste)
+ }
> colnames(resultKNNMDS) <- c("N","Accuracy")
> meanResultKNNMDS <- mean(resultKNNMDS[,2])
> results[["KNNMDS"]] <- resultKNNMDS
```

O resultado para este treinamento pode ser visto a seguir.

```
       N Accuracy
 [1,]  1     85.5
 [2,]  2     75.0
 [3,]  3     78.0
 [4,]  4     71.0
 [5,]  5     71.5
 [6,]  6     72.0
 [7,]  7     71.5
```

```
 [8,]  8     68.5
 [9,]  9     66.5
[10,] 10     67.5
```

3.7 Classificação utilizando Bayes e PCA

A regra de classificação de Bayes, que minimiza o risco de classificação, realiza a decisão se uma determinada amostra pertence ou não à uma classe, conforme à análise comparativa das probabilidades posteriores de pertencimento da mesma entre todas as outras. A amostra pertencerá à classe com maior probabilidade naquele ponto, se considerarmos que cada uma de suas n features equivale a um ponto em um sistema Rn.

A função utilizada para a classificação de Bayes foi a naive_bayes, do pacote naivebayes, pode ser vista à seguir.

```
> pred2 <- naive_bayes(x=xtreinoPCA, y=ytreino, useKernel =TRUE)
> predr <- predict(pred2,xtestePCA)
> resultBayesPCA <- checkAcc(predr,yteste)
```

3.8 Classificando com Bayes e MDS

A classificação com a função *naive_bayes* com o conjunto de classes reduzido com MDS utiliza do trecho de código a seguir.

```
> pred3 <- naive_bayes(x=xtreinoMDS, y=ytreino, useKernel =TRUE)
> predr <- predict(pred3,xtesteMDS)
> resultBayesMDS <- checkAcc(predr,yteste)
```

3.9 Resultados com Bayes

Comparando, portanto as duas classificações utilizando o classificador de Bayes, tanto para a redução de parêmetros através de PCA quanto para a utilizando MDS, temos os seguintes resultados:

O

```
      N Accuracy
PCA 125     62.5
MDS 114     57.0
```

4 Avaliação dos Resultados

Anteriormente, durante o desenvolviemento do trabalho, houve a tentativa de utilização de uma roina própria de implementação do classificador de bayes. A função implementada foi a seguinte:

```
> bayes <- function(xtreino, ytreino, nSamples, xteste, yteste) {
+
+     Mcov <- list()
+     nClass <- nrow(xtreino) / nSamples
+     classe <- 1
+     meansList <- list()
+     nFeatures <- ncol(xtreino)
+
+     for (i in seq(1,nrow(xtreino), nSamples)) {
+         Mcov[[classe]] <- cov(xtreino[i:(i+nSamples-1),])
+         meansList[[classe]] <- colMeans(xtreino[i:(nSamples-1),])
+         classe <- classe + 1
+     }
+
+     pdf <- matrix(nrow = nrow(xteste), ncol = nClass)
+
+     for (i in seq(nrow(xteste))) {
+         for (j in seq(nClass)) {
+             pdf[i,j] <- pdfnvar(xteste[i,],meansList[[j]],Mcov[[j]],nFeatures)
+         }
+     }
+
+     predResult <- matrix(nrow = nrow(pdf))
+     predResult <- apply(pdf, 1,which.max)
+
+     results <- list()
+     checkAcc <- predResult - yteste
+     win <- sum(checkAcc == 0)
+     predWin <- win / length(yteste) * 100
+
+     results["predResult"] <- predResult
+     results["predWin"] <- predWin
+
+
+     return(results)
```

```
+ }
>
```

Observando a implementação da função do classificador de Bayes, e da geração de pdfs para amostras multivariáveis, uma matriz de correlação invertível é gerada apenas se o número de linhas for menor ou igual ao número de colunas, única possibilidade que não gera uma matriz singular quando da execução da função *solve*. Como são 10 amostras de cada classe, e para o trabalho será utilizada 5 amostras para treino e 5 para testes, o número de features máximo que pode ser utilizado, desta forma, para o correto funcionamento do algoritmo do classificador de Bayes, é de 4 features.

Porém, a utilização de uma quantidade tão inferior de parâmetros compromete a classificação correta das amostras.

Um ponto de destaque observando os algoritmos é a diminuição da acurácia do algoritmo KNN conforme o número de vizinhos selecionados aumenta. Este comportamento parece ser contra-intuitivo, uma vez que o senso comum nos diz que quanto mais vizinhos idênticos a um determinado ponto, maior a confiabilidade do mesmo naquele resultado. Uma das explicações possíveis para tal é a possibilidade de haver sobreposição de amostras de treinamento dentro do espaço que as contém, o que nos indica que quanto amis vizinhos selecionarmos para confirmar a classe de uma amostra de teste, maiores as chances de selecionarmos pontos de classes erradas.